Multiple Complimentary Partnerships in Engineering Education

Keywords
Engineering education, Problem-based Learning, Software Engineering

Abstract
Requirements Engineering (RE) has been classed as a 'wicked' problem that is not well addressed by traditional formal education: studies show that the knowledge taught does not match the knowledge needed to be applied in daily work. Further studies suggest that the prescribed syllabus of formal courses is a major factor in the mismatch. This is the impetus for the review of the unit offered through the School of Engineering. It involves the development of a series of partnerships leading to a course that provides a solid foundation in subject matter while at the same time exposing students to inherent characteristics associated with real RE problems and the knowledge needed to solve them:

- *Industry/university partnership*: our discussions with employers show that course content and graduate grades are no longer a major issue in graduate employment scenarios. The focus is on how students transition to being professionals within industry. The aim is to provide a more authentic learning environment, one that more closely simulates the real world, through a Problem-based Learning (PBL) experience PBL encourages students to practice and develop the skills they will use as professional Software Engineers
- *Expert/novice partnership*: feedback from our students suggests they want increased exposure to real world problems. One impetus for the restructure of this unit is students wish to engage with the material on offer '*you need more practical application of the theory you teach.*' When the nature of the learning experience models work, students perceive themselves as apprentice professionals, and learning results from undertaking activities guided by an expert
- *Teaching/learning/discipline (engineering) partnership*: providing a well-balanced learning experience requires expertise both in the domain and in the theory of learning. This restructure provides an opportunity for reflection on teaching practice

This paper looks at the characteristics of these partnerships in the context of the redevelopment of the Requirements Engineering unit, and the issues raised by implementing PBL for learning basic disciplinary knowledge.

Jocelyn Armarego
Lecturer
School of Engineering
Murdoch University
Rockingham WA 6168
Ph: 9360 7118
Fax: 9360 7104
jocelyn@eng.murdoch.edu.au

Sally Clarke
Coordinator, Academic Staff Development
Teaching and Learning Centre
Murdoch University
Murdoch WA 6150
Ph: 9360 6418
Fax: 9310 4929
saclarke@central.murdoch.edu.au

Introduction

Software development has been regarded as a craft. The negative connotations of this label include an inability to consistently guarantee a quality product, fit for the purpose for which it was developed, produced on time and within budget. Statistics show that an unacceptably high percentage of software is under utilised once deployed, fails to be deployed once developed, or simply is not completed (eg according to a study of over 8,000 projects (Standish, 1995):16.2% were successful (on time, on budget, full functionality), 52.7% were challenged (over budget, time and fewer features), 31.1% were impaired (project was cancelled). These rates are not significantly different than when systems were built in the 70's and 80's (see (Mann, 1996) for a review of failure literature). Many of the shortfalls may be traced to deficiencies in formulating a description of the system to be developed. This phase of software development is often referred to as Requirements Engineering (RE), and is the target of this discussion.

One approach to addressing these issues has been to focus on the the engineering paradigm, prescribing detailed development and management processes and imposing quality standards on all aspects of the development cycle. (Royce, 1970)was the first to note explicitly that an engineering approach to software development was required, in the expectation that adherance to a defined, repeatable process will enhance quality. The underlying assumption is that the world works rationally and that therefore “good" software development is achieved by applying (from a choice of) scientific investigative techniques (Pfleeger, 1999).

This focus on engineering is mirrored in the education of software developers. Where two engineering of software programs at undergraduate level were accredited by IEAust (the Institution of Engineers Australia) in the mid-1990s (Melbourne, Murdoch), by 2002 this figure approached 20. A similar trend is being shown in the US, with an exponential growth in offerings of undergraduate software engineering degrees.

More recent work (Maiden & Gizikis, 2001; Nguyen & Swatman, 2000) argues such an approach should be regarded as flawed in that it is based on fundamentally wrong ideas regarding the successful development of software. Unlike the engineering and manufacturing metaphors used to drive these views, software development is dominated by human cognition. Software is a collaborative invention: software development is an exploratory and self-correcting dialogue (Bach, 1999).

This alternate perspective suggests that Requirements Engineers are not given problems, rather they construct them (Visser, 1992). This construction is thought to be insight-driven and fundamentally opportunistic (Carroll & Swatman, 1999; Guindon, 1989). The process of RE is seen as one of knowledge discovery (Guindon, 1989) facilitated by opportunistic behaviour (Guindon, 1990; Visser, 1992). The RE builds fragments of understanding of the problem validated and consolidated through the traversal of layers, collecting more areas and information at each, adding detail and richness to the mental model of the problem situation (Batra & Davis, 1992). Participants in the process must remain sensitive to these progressive modifications (Gigch, 2000) which lead not to a problem-solution, but to an ‘evolved fit’ acceptable to all stakeholders within the problem space.

Educating software developers

RE education has been referred to as an "educational dilemma"' (Macauley & Mylopoulos, 1995). The dilemma is to provide the student with a solid foundation in subject matter while at the same time exposing the student to the inherent characteristics associated with real requirements problems and the knowledge required to solve them. These characteristics have been summarised (Bubenko, 1995) as:

- complexity is augmented rather than reduced with increased understanding of the initial problem
- metacognitive strategies are fundamental to the process
- problem-solving needs a rich background of knowledge and intuition to operate effectively
- a breadth of experience is necessary so that similarities and differences with past strategies are used to deal with new situations.

However, approaches to training Requirements Engineers based on traditional learning models tend to focus on technical knowledge, and are based largely on notations and prescribed processes. The mismatch between 'modelled' and actual behaviour, a discrepancy between theory and practice (Glass, 1995) is supported in the literature on expert behaviour (Robillard, 1999; Visser & Hoc, 1990). Experts don't do in practice what they say the do because their own plans are cognitively more cost effective and flexible, allowing for creativity and opportunism.

The educational dilemma now becomes one to provide an educational base that enables software developers to both create and engineer the systems they build: to be adaptable to the changing environment that is inevitable in their chosen discipline.

The Murdoch context

Murdoch University provides a four year undergraduate Software Engineering (SE) degree (amongst others). Requirements Engineering is the first of the core SE units, offered in semester 1 of the second year of study. The implication of this is that students have been immersed (except for the foundation unit[1] in Semester 1 of first year) in a scientific/ engineering paradigm: laboratory procedure, repeatability of experimentation and rigour in mathematics are the key learning objectives of the introductory units taken. As these are common to all engineering flavours (eg Software, Instrumentation, Renewable, etc), the particular needs of each discipline are ignored. Students emerge with a disciplined, engineering frame of mind.

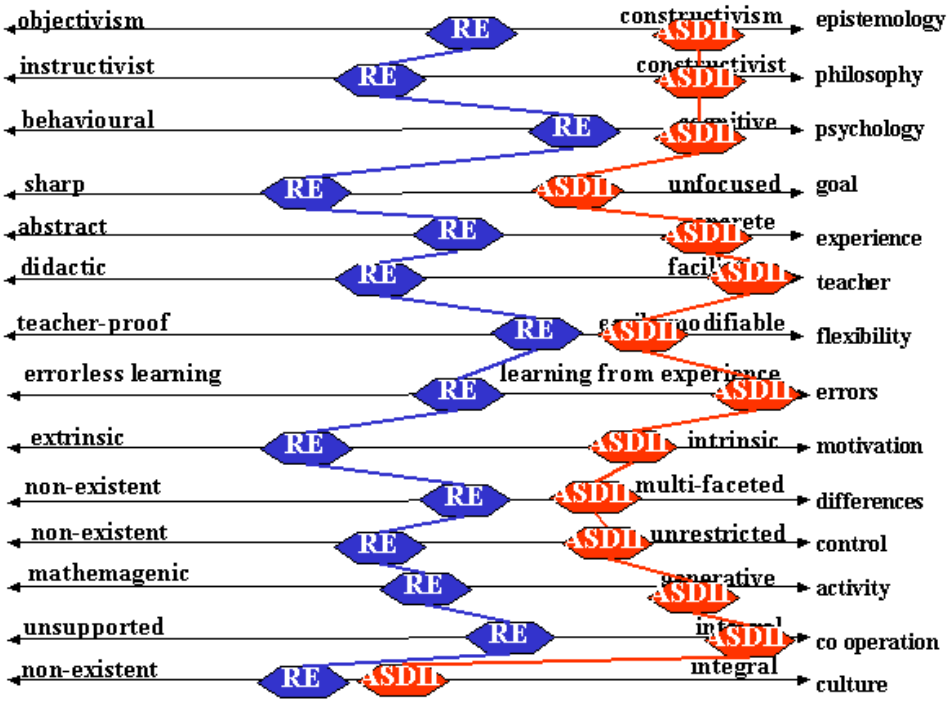
RE provides a contrast that some students find difficult to assimilate. Although due process and procedure has its place, the focus of the unit is on divergence thinking and the development and evaluation of alternatives. Students come to the unit with some competence in programming from their first year studies. In Requirements Engineering they are asked to ignore the (coding) solution to a situation presented, and to explore and then formulate the problem itself. Experience in teaching the course has shown that students' expectations are challenged:

[1] This is a compulsory unit in university skills aimed at easing the transition to tertiary study for both school leavers and mature age students. A multi-disciplinary approach is taken, although the version available to engineering students is co ordinated from the School of Engineering for all (including non-engineering) students enrolled at Rockingham.

- they expect there to exist a definitive solution to the problems with which they are presented (à la science/mathematics)
- they expect to define the problems only in terms of the programming language with which they are familiar (currently Java)
- they expect a 'normal' (and fundamentally competitive) class environment to exist. Students are initially confused in differentiating between the amount of collaboration required, and collusion
- they expect their 'wild ideas' to be laughed at and ultimately rejected, and therefore are inhibited in expressing them.

The course material has, since its inception in 1999, been taught in workshop mode. All material is available online, so lectures and tutorials are replaced by discussion, exercises and group evaluation of alternatives presented. While this could be classed as successful, if based on academic results and student evaluation of teaching, a review of the course based on Reeves' 14 dimensions for the evaluation of computer-based education (Reeves, 1997) showed that there was a reasonably high level of teacher direction within the course. This was borne out by student response to changes made to a follow-on course (Advanced Software Design II (ASDII)) presented in a learner centred mode. The expectations noted above were still evident (Figure 1 shows the evaluation results for both courses).

Figure 1 Evaluation of RE and ASDII based on Reeves (1997)



References

Bach, J. (1999). Reframing requirements analysis. *IEEE Computer, 32*(2), 120-122.

Batra, D., & Davis, J. G. (1992). Conceptual data modelling in database design: similarities and differences between expert and novice designers. *International Journal of Man-Machine Studies, 37*, 83-101.

Bubenko, J. (1995). *Challenges in Requirements Engineering: keynote address.* Paper presented at the RE'95: Second IEEE International Symposium on Requirements Engineering, York (UK).

Carroll, J. M., & Swatman, P. A. (1999). *Opportunism in the Requirements Engineering process* (School of Management Information Systems Working Paper 1999/02): Deakin University, Australia.

Gigch, J. P. v. (2000). Metamodelling and problem solving. *Journal of Applied Systems Studies, 1*(2), 327-336.

Glass, R. L. (1995). A theory about software's practice (Editor's Corner). *Journal of Systems and Software, 28*, 187-188.

Guindon, R. (1989). The process of knowledge discovery in system design. In G. Salvendy & M. J. Smith (Eds.), *Designing and Using Human-Computer Interfaces and Knowledge Based Systems* (pp. 727-734). Amsterdam: Elsevier.

Guindon, R. (1990). Knowledge exploited by experts during software systems design. *International Journal of Man-Machine Studies, 33*, 279-304.

Macauley, L., & Mylopoulos, J. (1995). Requirements Engineering: an educational dilemma. *Automated Software Engineering, 4*(2), 343-351.

Maiden, N., & Gizikis, A. (2001). Where do requirements come from? *IEEE Software, 18*(5), 10-12.

Mann, J. (1996). *The Role of Project Escalation in Explaining Runaway Information Systems Development Projects: A Field Study.* Georgia State University.

Nguyen, L., & Swatman, P. A. (2000). *Complementary Use of ad hoc and post hoc Design Rationale for Creating and Organising Process Knowledge.* Paper presented at the Proceedings of the Hawaii International Conference on System

Sciences HICSS-33, Maui (Hawaii).

Pfleeger, S. L. (1999). Albert Einstein and empirical software engineering. *IEEE Computer, 32*(10), 32-37.

Reeves, T. C. (1997). *A Model of the effective dimensions of interactive learning on the World Wide Web.* Paper presented at the Proceedings of Interaktiivinen Teknologia Koulutuksessa (ITK '97), Hameenlinna (Finland).

Robillard, P. N. (1999). The role of knowledge in software development. *Communications of the ACM, 42*(1), 87-92.

Royce, W. W. (1970). *Managing the development of large software systems: concepts and techniques.* Paper presented at the IEEE WESCON.

Standish. (1995). *Most Programming Projects Are Late*. West Yarmouth (MA): Standish Group.

Visser, W. (1992). Designers' activities examined at three levels: organisation strategies and problem-solving processes. *Knowledge-Based Systems, 5*(1), 92-104.

Visser, W., & Hoc, J. (1990). Expert software design strategies. In J. M. Hoc & T. R. G. Green & S. R & G. D. J (Eds.), *Psychology of Programming* (pp. 235-247). San Diego (CA): Academic Press.